APPENDIX A

```
// SolverDLL.cpp : implementation file
//

#include "stdafx.h"
#include "SecretSet.h"
#include "SolverDLL.h"
#include "ProgDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSolverDLL

CSolverDLL::CSolverDLL()
{
}

CSolverDLL::~CSolverDLL()
{
}


BEGIN_MESSAGE_MAP(CSolverDLL, CWnd)
        //{{AFX_MSG_MAP(CSolverDLL)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CSolverDLL message handlers

void CSolverDLL::OnRun()
{
        float      CurTime = 0.0f;
        float      EndofDay = 23.99f;
        int          i, assigned;
        int          nextTask;
        CTRRes  *Rptr;
        CAssign *Aptr;
        int          tmpfnd;
```

```
	char*	pFileName = _T("marktime.txt");
	CTime		TimeNow;
	CString buffer;

	CFile rfile( pFileName, CFile::modeCreate | CFile::modeWrite );
	buffer.Format("Start Getting information from DB:
\t"+TimeNow.GetCurrentTime().Format("%H:%M:%S\n"));
	rfile.Write(buffer, buffer.GetLength());

	CurTime = __max(STARTING-ELAPSE,0);
	posEvent = NULL;
	posAssign = NULL;
	posQList = NULL;
	resultQ = NULL;
	EventList = NULL;
	ResList = NULL;
	Shifts = NULL;
	update_value += 1.0f;
	Progress.SetPos((int) update_value);
	Progress.AddtoList("Getting Resource Information...");
//	MessageBox("Get Res Info - 3","Mark",MB_OK);
	GetResInfo();
	Progress.AddtoList("Getting Task Information...");
//	MessageBox("Get Task Info - 4","Mark",MB_OK);
	GetTaskInfo();
	Progress.AddtoList("Getting Queue Information...");
	GetEventList();

	buffer.Format("Start Time of Algorithm:
\t"+TimeNow.GetCurrentTime().Format("%H:%M:%S\n"));
	rfile.Write(buffer, buffer.GetLength());

	Progress.AddtoList("Starting LF Algorithm...");

	CurTime = GetNextTime(CurTime);
	while (CurTime <= EndofDay)
	{
		buffer.Format("Looking at Time Period: "+UnConvertTime(CurTime));
		Progress.AddtoList(buffer);
		PrintQs(CurTime);
		// Initialize work array
		for (i = 0; i < numTask; i++)
		{
			QList[i].m_done = 0.0f;
			QList[i].m_go = 0;
```

```
        QList[i].m_assigned = 0;
        if (QList[i].teamList != NULL)
        {
                CTeam *tptr;
                tptr = QList[i].teamList;
                while (tptr != NULL)
                {
                        tptr->rptr = NULL;
                        tptr = tptr->next;
                }
        }
        // Count the num of resources currently assigned
        Rptr = QList[i].resList;
        while (Rptr!= NULL)
        {
                Aptr = Rptr->assigned;
                tmpfnd = FALSE;
                while ((Aptr != NULL) && !tmpfnd)
                {
                        if (Aptr->m_end_tim > CurTime)
                        {
                                tmpfnd = TRUE;
                                QList[i].m_assigned++;
                                break;
                        }
                        Aptr = Aptr->next;
                }
                Rptr = Rptr->next;
        }
}
nextTask = 0;
while (nextTask < numTask)
{
        CalculateWork(CurTime);
        qsort(QList,numTask, sizeof(CQelement), dcompare);
        nextTask = GetNextTask(CurTime);
        if (nextTask < numTask)
        {
                assigned = AssignRes(nextTask, CurTime);
                if (!assigned)
                        QList[nextTask].m_go = 1;
        }
}

CurTime = GetNextTime(CurTime);
```

```
                CalculateQ(CurTime);
                update_value += (10.0f/(int)((EndofDay-STARTING)/ELAPSE));
                Progress.SetPos((int)update_value);
        }

        buffer.Format("Start updating DB:
\t"+TimeNow.GetCurrentTime().Format("%H:%M:%S\n"));
        rfile.Write(buffer, buffer.GetLength());

        Progress.AddtoList("Update Tables...");
        SaveAssign();
        Progress.AddtoList("Cleanup Memory...");
        CleanupMem();

        buffer.Format("End updating DB:
\t"+TimeNow.GetCurrentTime().Format("%H:%M:%S\n"));
        rfile.Write(buffer, buffer.GetLength());
        rfile.Close();

}


int dcompare(const void *p, const void *q)
{
        // Compares the Rxs in each task
/*      if ((ListTask(p)->m_Rxs - ListTask(q)->m_Rxs) > 0)
                return(-1);
        else
                if ((ListTask(p)->m_Rxs - ListTask(q)->m_Rxs) < 0)
                        return(1);
                else
                        return(0);
*/
        // Compares the work in each task
        if ((ListTask(p)->m_work - ListTask(q)->m_work) > 0)
                return(-1);
        else
                if ((ListTask(p)->m_work - ListTask(q)->m_work) < 0)
                        return(1);
                else
                        return(0);

}
```

```
void CSolverDLL::GetTaskInfo()
{
	CTaskFlow	Flow(NULL);
	CToList			*curtoptr;
	CRelement	*Rptr;
	CTRRes			*curRptr;
	CResAvail	RAvailList(NULL);
	CAllTask			TaskSet(NULL);

	RAvailList.m_ConnectStr = DB_CONNECT;
	// Opens the first instance of the RAvailList for Requery
	RAvailList.m_day_Param = DAYCDE;
	RAvailList.m_location_Param = 1;
	RAvailList.m_Pharmacy_Param = PHARMACY;
	try
	{
		RAvailList.Open();
	}
	catch(CDBException *e)
	{
		e->ReportError(MB_ICONEXCLAMATION);
		CleanupMem();
		exit;
	}
	update_value += 1.0f;
	Progress.SetPos((int) update_value);

	// Opens the first instance of the Flow for Requery
	Flow.m_ConnectStr = DB_CONNECT;
	Flow.m_Pharmacy_Param = PHARMACY;
	Flow.m_location_Param = 1;
	try
	{
		Flow.Open();
	}
	catch(CDBException *e)
	{
		e->ReportError(MB_ICONEXCLAMATION);
		CleanupMem();
		exit;
	}
	update_value += 1.0f;
	Progress.SetPos((int) update_value);

	Progress.AddtoList("Getting Task List...");
```

```
// List all the task in the system
TaskSet.m_ConnectStr = DB_CONNECT;
TaskSet.m_day_Param = DAYCDE;
TaskSet.m_Pharmacy_Param = PHARMACY;
try
{
        TaskSet.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
}

int i = 0;
while ( !TaskSet.IsEOF( ) )
{
// Creates the Queues for each of the tasks
        QList[i].m_location_task_id = TaskSet.m_location_task_id;
        QList[i].m_Rxs = 0.0f;
        QList[i].m_RealQ = 0.0f;
        QList[i].m_done = 0.0f;
        QList[i].m_assigned = 0;
        QList[i].m_work = 0.0f;
        QList[i].m_max_resource_qty = TaskSet.m_max_resource_qty;
        QList[i].minList = NULL;
        QList[i].maxList = NULL;
        QList[i].ratList = NULL;
        QList[i].teamList = NULL;
        QList[i].resList = NULL;
        QList[i].m_go = 0;
        QList[i].m_start_tim = ConvertDBTime(TaskSet.m_start_tim);
        QList[i].m_end_tim = ConvertDBTime(TaskSet.m_end_tim);

        // Gets the resource list for the task in order of perference
        RAvailList.m_day_Param = DAYCDE;
        RAvailList.m_location_Param = TaskSet.m_location_task_id;
        try
        {
                RAvailList.Requery();
        }
        catch(CDBException *e)
        {
                e->ReportError(MB_ICONEXCLAMATION);
```

```
        CleanupMem();
        exit;
}

while (!RAvailList.IsEOF())
{
        CTRRes *tmptr = new CTRRes();
        tmptr->m_total_time = 0.0f;
        if (RAvailList.m_hourly_rate <= 0)
                tmptr->m_hourly_rate = (float)TaskSet.m_hourly_rate;
        else
                tmptr->m_hourly_rate = (float)RAvailList.m_hourly_rate;
        tmptr->assigned = NULL;
        tmptr->next = NULL;
        Rptr = ResList;
        if (Rptr != NULL)
        {
                while (Rptr != NULL)
                {
                        if (Rptr->m_resource_id == RAvailList.m_resource_id)
                                break;
                        Rptr = Rptr->next;
                }
                if (Rptr->m_resource_id == RAvailList.m_resource_id)
                        tmptr->resource = Rptr;
        }
        if (QList[i].resList == NULL)
                QList[i].resList = tmptr;
        else
                curRptr->next = tmptr;
        curRptr = tmptr;
        RAvailList.MoveNext();
}

//Gets Flow Data for taskList
QList[i].flowto = NULL;
Flow.m_location_Param = TaskSet.m_location_task_id;
try
{
        Flow.Requery();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
```

```
                exit;
        }

        curtoptr = NULL;
        while (!Flow.IsEOF())
        {
                CToList *toptr = new CToList();
                toptr->m_to_task_id = Flow.m_to_task_id;
                toptr->m_allocation_pct = Flow.m_allocation_pct;
                toptr->next = NULL;
                if (QList[i].flowto == NULL)
                        QList[i].flowto = toptr;
                else
                        curtoptr->next = toptr;
                curtoptr = toptr;
                Flow.MoveNext();
        }

        i++;
        TaskSet.MoveNext();
        update_value += 10.0f/numTask;
        Progress.SetPos((int) update_value);
}
TaskSet.Close();
RAvailList.Close();
Flow.Close();

Progress.AddtoList("Getting Min/Max Constraints...");
/* Gets a list of the minimum times for each
        resource_type_cde and task combination */
CMinMax MinMax(NULL);

MinMax.m_ConnectStr = DB_CONNECT;
MinMax.m_strSort = _T("location_task_id, resource_type_cde");
MinMax.m_Pharmacy_Param = PHARMACY;
MinMax.m_type_Param = _T("min constraint");
try
{
        MinMax.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
```

```
}

while (!MinMax.IsEOF())
{
        for (i = 0; i < numTask; i++)
        {
                if (QList[i].m_location_task_id == MinMax.m_location_task_id)
                {
                        CMMList *mlist = new CMMList();
                        mlist->m_resource_type_cde = MinMax.m_resource_type_cde;
                        mlist->m_time_amt = MinMax.m_time_amt/60.0f;
                        mlist->next = NULL;
                        if (QList[i].minList == NULL)
                                QList[i].minList = mlist;
                        else
                        {
                                CMMList *ptr;
                                ptr = QList[i].minList;
                                while (ptr->next != NULL)
                                        ptr = ptr->next;
                                ptr->next = mlist;
                        }
                        break;
                }
        }
        MinMax.MoveNext();
}
update_value += 1.0f;
Progress.SetPos((int) update_value);

/* Gets a list of the maximum times for each
        resource_type_cde and task combination */
MinMax.m_type_Param = _T("max constraint");
try
{
        MinMax.Requery();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
}

while (!MinMax.IsEOF())
```

```
{
        for (i = 0; i < numTask; i++)
        {
                if (QList[i].m_location_task_id == MinMax.m_location_task_id)
                {
                        CMMList *mlist = new CMMList();
                        mlist->m_resource_type_cde = MinMax.m_resource_type_cde;
                        mlist->m_time_amt = MinMax.m_time_amt/60.0f;
                        mlist->next = NULL;
                        if (QList[i].maxList == NULL)
                                QList[i].maxList = mlist;
                        else
                        {
                                CMMList *ptr;
                                ptr = QList[i].maxList;
                                while (ptr->next != NULL)
                                        ptr = ptr->next;
                                ptr->next = mlist;
                        }
                        break;
                }
        }
        MinMax.MoveNext();
}
MinMax.Close();

update_value += 1.0f;
Progress.SetPos((int) update_value);
Progress.AddtoList("Getting Team Constraints...");
// Gets a list of the resource ratios for each task
CRatioSet      RatSet(NULL);

RatSet.m_ConnectStr = DB_CONNECT;
RatSet.m_Pharmacy_Param = PHARMACY;
RatSet.m_type_Param = _T("ratio constraint");
RatSet.m_strSort = _T("location_task_id, res1, res2");
try
{
        RatSet.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
```

```
}

while (!RatSet.IsEOF())
{
        for (i = 0; i < numTask; i++)
        {
                if (QList[i].m_location_task_id == RatSet.m_location_task_id)
                {
                        CRatio *rlist = new CRatio();
                        rlist->m_res1 = RatSet.m_res1;
                        rlist->m_res2 = RatSet.m_res2;
                        rlist->m_ratio1 = 1;
                        rlist->m_ratio2 = RatSet.m_ratio_amt;
                        rlist->next = NULL;
                        if (QList[i].ratList == NULL)
                                QList[i].ratList = rlist;
                        else
                        {
                                CRatio *ptr;
                                ptr = QList[i].ratList;
                                while (ptr->next != NULL)
                                        ptr = ptr->next;
                                ptr->next = rlist;
                        }
                        break;
                }
        }
        RatSet.MoveNext();
}
RatSet.Close();
update_value += 1.0f;
Progress.SetPos((int) update_value);

// Set up Teams templates
for (i = 0; i < numTask; i++)
{
        if (QList[i].ratList != NULL)
        {
                CRatio  *ratptr;
                CTeam           *lastteam;
                CRatTeam     *resultList;
                CRatTeam *lastone;

                ratptr = QList[i].ratList;
                CRatTeam *rlptr = new CRatTeam();
```

```
rlptr->m_resource_type_cde = ratptr->m_res1;
rlptr->m_count = ratptr->m_ratio1;
rlptr->next = new CRatTeam();
resultList = rlptr;
rlptr = rlptr->next;
rlptr->m_resource_type_cde = ratptr->m_res2;
rlptr->m_count = ratptr->m_ratio2;
rlptr->next = NULL;
lastone = rlptr;
ratptr = ratptr->next;
while (ratptr != NULL)
{
        rlptr = resultList;
        while (rlptr != NULL)
        {
                if (rlptr->m_resource_type_cde == ratptr->m_res1)
                {
                        int totalres = rlptr->m_count *ratptr->m_ratio1;
                        int gcdresult = gcd(rlptr->m_count,
ratptr->m_ratio1);
                        totalres = totalres/gcdresult;
                        int multby = totalres/rlptr->m_count;
                        rlptr = resultList;
                        while (rlptr != NULL)
                        {
                                rlptr->m_count *= multby;
                                rlptr = rlptr->next;
                        }
                        multby = totalres/ratptr->m_ratio1;
                        CRatTeam *rlptr = new CRatTeam();
                        rlptr->m_count = ratptr->m_ratio2*multby;
                        rlptr->m_resource_type_cde = ratptr->m_res2;
                        rlptr->next = NULL;
                        lastone->next = rlptr;
                        lastone = rlptr;
                        break;
                }
                rlptr = rlptr->next;
        }
        if (rlptr == NULL)
        {
                rlptr = resultList;
                while (rlptr != NULL)
                {
                        if (rlptr->m_resource_type_cde == ratptr->m_res2)
```

```
                                        {
                                                int totalres = rlptr->m_count
*ratptr->m_ratio2;
                                                int gcdresult = gcd(rlptr->m_count,
ratptr->m_ratio2);
                                                totalres = totalres/gcdresult;
                                                int multby = totalres/rlptr->m_count;
                                                rlptr = resultList;
                                                while (rlptr != NULL)
                                                {
                                                        rlptr->m_count *= multby;
                                                        rlptr = rlptr->next;
                                                }
                                                multby = totalres/ratptr->m_ratio2;
                                                CRatTeam *rlptr = new CRatTeam();
                                                rlptr->m_count = ratptr->m_ratio1*multby;
                                                rlptr->m_resource_type_cde =
ratptr->m_res1;
                                                rlptr->next = NULL;
                                                lastone->next = rlptr;
                                                lastone = rlptr;
                                                break;
                                        }
                                        rlptr = rlptr->next;
                                }
                        }
                }
                rlptr = resultList;
                lastteam = QList[i].teamList;
                while (rlptr != NULL)
                {
                        for (int w = 0; w < rlptr->m_count; w++)
                        {
                                CTeam *teamptr = new CTeam();
                                teamptr->cont_prev = FALSE;
                                teamptr->m_resource_type_cde =
rlptr->m_resource_type_cde;
                                teamptr->rptr = NULL;
                                teamptr->next = NULL;
                                if (lastteam == NULL)
                                        QList[i].teamList = teamptr;
                                else
                                        lastteam->next = teamptr;
                                lastteam = teamptr;
                        }
```

```
                        rlptr = rlptr->next;
                    }
                    rlptr = resultList;
                    if (resultList != NULL)
                    {
                        while (resultList->next != NULL)
                        {
                            rlptr = rlptr->next;
                            delete resultList;
                            resultList = rlptr;
                        }
                        delete resultList;
                    }
                }
                update_value += 10.0f/numTask;
                Progress.SetPos((int) update_value);
            }
        }

        void CSolverDLL::GetResInfo()
        {
            CRelement    *curResptr;
            CResAll                 ResListSet(NULL);
            CShiftSet    ShiftList(NULL);
            CShiftID                *curptr;
            CTimePer     *curperiodptr;
            CNotAvail    OutDay(NULL);
            int                     curID;
            float                   how_much;

            // Gets a list of shifts
            Progress.AddtoList("Getting Shift Information...");
            ShiftList.m_ConnectStr = DB_CONNECT;
            ShiftList.m_Pharmacy_Param = PHARMACY;
            ShiftList.m_day_Param = DAYCDE;
            try
            {
                ShiftList.Open();
            }
            catch(CDBException *e)
            {
                e->ReportError(MB_ICONEXCLAMATION);
                CleanupMem();
                exit;
            }
```

```
curID = -10;
update_value += 1.0f;
Progress.SetPos((int) update_value);
while (!ShiftList.IsEOF())
{
        // Create a linked list of the shifts for faster access
        if (curID != ShiftList.m_Location_Shift_Break_shift_id)
        {
                CShiftID *shiftptr = new CShiftID();
                shiftptr->m_shift_id = ShiftList.m_Location_Shift_Break_shift_id;
                shiftptr->next = NULL;
                shiftptr->m_period = NULL;
                if (curID == -10)
                        Shifts = shiftptr;
                else
                        curptr->next = shiftptr;
                curID = ShiftList.m_Location_Shift_Break_shift_id;
                curptr = shiftptr;
                curperiodptr = NULL;
        }
        if (ShiftList.m_Location_Shift_Break_start_tim !=
ShiftList.m_Location_Shift_Break_end_tim)
        {
                CTimePer *periodptr = new CTimePer();
                periodptr->m_start_tim =
ConvertDBTime(ShiftList.m_Location_Shift_Break_start_tim);
                periodptr->m_end_tim =
ConvertDBTime(ShiftList.m_Location_Shift_Break_end_tim);
                periodptr->next = NULL;
                if (curperiodptr == NULL)
                        curptr->m_period= periodptr;
                else
                        curperiodptr->next = periodptr;
                curperiodptr = periodptr;
        }
        ShiftList.MoveNext();
}
ShiftList.Close();

update_value += 1.0f;
Progress.SetPos((int)update_value);
Progress.AddtoList("Getting Resource List...");

// Gets a list of all the resources
ResListSet.m_ConnectStr = DB_CONNECT;
```

```
ResListSet.m_day_Param = DAYCDE;
ResListSet.m_Pharmacy_Param = PHARMACY;
try
{
        ResListSet.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
}
update_value += 1.0f;
Progress.SetPos((int)update_value);

CSaveSet                Assignments(NULL);

Assignments.m_ConnectStr = DB_CONNECT;
try
{
        Assignments.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
}


how_much = update_value+10.0f;
while (!ResListSet.IsEOF())
{
        CRelement *Rptr = new CRelement();
        Rptr->m_resource_id = ResListSet.m_resource_id;
        Rptr->m_avail_time = ConvertDBTime(ResListSet.m_start_tim);
        Rptr->m_not_avail = 24.0f;
        Rptr->m_end_time = ConvertDBTime(ResListSet.m_end_tim);
        Rptr->m_cost = ResListSet.m_cost+1;
        Rptr->m_calculate = TRUE;
        Rptr->m_resource_type_cde = ResListSet.m_resource_type_cde;
        Rptr->m_last_task = -99;
        Rptr->breaks = NULL;
        Rptr->nAvail = NULL;
        if (Shifts != NULL)
```

```
{
        curptr = Shifts;
        while (curptr->next != NULL)
        {
                if (curptr->m_shift_id == ResListSet.m_shift_id)
                        break;
                curptr = curptr->next;
        }
        if (curptr->m_shift_id == ResListSet.m_shift_id)
        {
                Rptr->breaks = curptr;
                // spit out lunches and breaks
                CTimePer *perptr;

                perptr = curptr->m_period;
                Rptr->m_not_avail = curptr->m_period->m_start_tim;
                while (perptr != NULL)
                {
                        if (Assignments.CanAppend() >0)
                        {
                                Assignments.AddNew();
                                Assignments.m_location_task_id = BREAKID;
                                Assignments.m_resource_id =
Rptr->m_resource_id;
                                Assignments.m_scenario_id = SCENARIO;
                                Assignments.m_start_time =
UnConvertTime1(perptr->m_start_tim);
                                Assignments.m_end_time =
UnConvertTime1(perptr->m_end_tim);
                                Assignments.Update();
                        }
                        perptr = perptr->next;
                }
        }
        else
                Rptr->m_not_avail = Rptr->m_end_time;
}
Rptr->next = NULL;
if (ResList == NULL)
        ResList = Rptr;
else
        curResptr->next = Rptr;
curResptr = Rptr;

ResListSet.MoveNext();
```

```
        update_value += (10.0f/500.0f);
        update_value = __min(how_much,update_value);
        Progress.SetPos((int) update_value);
}
ResListSet.Close();
Assignments.Close();
update_value = how_much;
Progress.SetPos((int) update_value);
Progress.AddtoList("Getting Exception Information...");

// Get list of exceptions for resource
CString buffer;

buffer = _T("(e.resource_id = r.resource_id) and \
        (r.location_nbr = ?) and (e.dte = ");
buffer += RUN_DATE_TIME.Format("'%m/%d/%y')");
OutDay.m_strFilter = _T(buffer);
OutDay.m_strSort = _T("e.resource_id");
OutDay.m_ConnectStr = DB_CONNECT;
OutDay.m_Pharmacy_Param = PHARMACY;
try
{
        OutDay.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
}
update_value += 1.0f;
Progress.SetPos((int) update_value);
how_much = update_value + 3.0f;

Assignments.m_ConnectStr = DB_CONNECT;
try
{
        Assignments.Open();
}
catch(CDBException *e)
{
        e->ReportError(MB_ICONEXCLAMATION);
        CleanupMem();
        exit;
}
```

```
while (!OutDay.IsEOF())
{
        if (Assignments.CanAppend() >0)
        {
                Assignments.AddNew();
                Assignments.m_location_task_id = OutDay.m_location_task_id;
                Assignments.m_resource_id = OutDay.m_resource_id;
                Assignments.m_scenario_id = SCENARIO;
                Assignments.m_start_time = OutDay.m_start_tim;
                Assignments.m_end_time = OutDay.m_end_tim;
                Assignments.Update();
        }
        curResptr = ResList;
        while (curResptr != NULL)
        {
                if (curResptr->m_resource_id == OutDay.m_resource_id)
                {
                        CTimePer *nhptr = new CTimePer();
                        nhptr->m_end_tim = ConvertDBTime(OutDay.m_end_tim);
                        nhptr->m_start_tim = ConvertDBTime(OutDay.m_start_tim);
                        nhptr->next = NULL;
                        if (curResptr->nAvail == NULL)
                                curResptr->nAvail = nhptr;
                        else
                        {
                                CTimePer *nAptr;
                                nAptr = curResptr->nAvail;
                                while (nAptr->next != NULL)
                                        nAptr = nAptr->next;
                                nAptr->next = nhptr;
                        }
                        break;
                }
                curResptr = curResptr->next;
        }
        OutDay.MoveNext();
        update_value += (3.0f/50.0f);
        update_value = __min(how_much,update_value);
        Progress.SetPos((int) update_value);
}
OutDay.Close();
Assignments.Close();
update_value = how_much;
Progress.SetPos((int) update_value);
}
```

```
void CSolverDLL::CleanupMem()
{
	//Free memory from the TaskList
	CTRRes		*Rptr;
	CToList		*Fptr;
	CRatio		*RRptr;
	CMMList		*Mptr;
	CTeam			*TTptr;
	CAssign		*Aptr;

	for (int i = 0; i < numTask; i++)
	{
		if (QList[i].resList != NULL)
		{
			Rptr = QList[i].resList;
			while (QList[i].resList->next != NULL)
			{
				//Free memory from Assigned List
				if (Rptr->assigned != NULL)
				{
					Aptr = Rptr->assigned;
					while (Rptr->assigned->next != NULL)
					{
						Aptr = Aptr->next;
						delete Rptr->assigned;
						Rptr->assigned = Aptr;
					}
					delete Rptr->assigned;
				}
				Rptr = Rptr->next;
				delete QList[i].resList;
				QList[i].resList = Rptr;
			}
			if (Rptr->assigned != NULL)
			{
				Aptr = Rptr->assigned;
				while (Rptr->assigned->next != NULL)
				{
					Aptr = Aptr->next;
					delete Rptr->assigned;
					Rptr->assigned = Aptr;
				}
				delete Rptr->assigned;
			}
			delete QList[i].resList;
```

```
        QList[i].resList = NULL;
}

if (QList[i].flowto != NULL)
{
        Fptr = QList[i].flowto;
        while (QList[i].flowto->next != NULL)
        {
                Fptr = Fptr->next;
                delete QList[i].flowto;
                QList[i].flowto = Fptr;
        }
        delete QList[i].flowto;
        QList[i].flowto = NULL;
}

if (QList[i].ratList != NULL)
{
        RRptr = QList[i].ratList;
        while (QList[i].ratList->next != NULL)
        {
                RRptr = RRptr->next;
                delete QList[i].ratList;
                QList[i].ratList = RRptr;
        }
        delete QList[i].ratList;
}

if (QList[i].teamList != NULL)
{
        TTptr = QList[i].teamList;
        while (QList[i].teamList->next != NULL)
        {
                TTptr = TTptr->next;
                delete QList[i].teamList;
                QList[i].teamList = TTptr;
        }
        delete QList[i].teamList;
}

if (QList[i].minList != NULL)
{
        Mptr = QList[i].minList;
        while (QList[i].minList->next != NULL)
        {
```

```
                Mptr = Mptr->next;
                delete QList[i].minList;
                QList[i].minList = Mptr;
            }
            delete QList[i].minList;
        }

        if (QList[i].maxList != NULL)
        {
            Mptr = QList[i].maxList;
            while (QList[i].maxList->next != NULL)
            {
                Mptr = Mptr->next;
                delete QList[i].maxList;
                QList[i].maxList = Mptr;
            }
            delete QList[i].maxList;
        }
        update_value += (3.0f/numTask);
        Progress.SetPos((int) update_value);
}
delete [] QList;


//Free memory from QueueList
CQResultList   *qptr;

if (resultQ != NULL)
{
        qptr = resultQ;
        while (resultQ->next != NULL)
        {
            qptr = qptr->next;
            delete resultQ;
            resultQ = qptr;
        }
        delete resultQ;
}

//Free memory from EventList
CTelement      *Tptr;

if (EventList != NULL)
{
        Tptr = EventList;
```

```
        while (EventList->next != NULL)
        {
                Tptr = Tptr->next;
                delete EventList;
                EventList = Tptr;
        }
        delete EventList;
}
//Free memory from linked lists for ResList
CRelement     *fromptr;
CTimePer              *nAptr;

if (ResList != NULL)
{
        fromptr = ResList;
        while (ResList->next != NULL)
        {
                if (ResList->nAvail != NULL)
                {
                        nAptr = ResList->nAvail;
                        while (ResList->nAvail->next != NULL)
                        {
                                nAptr = nAptr->next;
                                delete ResList->nAvail;
                                ResList->nAvail = nAptr;
                        }
                        delete ResList->nAvail;
                }
                fromptr = fromptr->next;
                delete ResList;
                ResList = fromptr;
        }
        if (ResList->nAvail != NULL)
        {
                nAptr = ResList->nAvail;
                while (ResList->nAvail->next != NULL)
                {
                        nAptr = nAptr->next;
                        delete ResList->nAvail;
                        ResList->nAvail = nAptr;
                }
                delete ResList->nAvail;
        }
        delete ResList;
}
```

```
update_value += 1.0f;
Progress.SetPos((int) update_value);


//Free memory from linked lists for Shift Definitions
CShiftID            *Sptr;
CTimePer     *Pptr;


if (Shifts != NULL)
{
        Sptr = Shifts;
        while (Shifts->next != NULL)
        {
                if (Shifts->m_period != NULL)
                {
                        Pptr = Shifts->m_period;
                        while (Shifts->m_period->next != NULL)
                        {
                                Pptr = Pptr->next;
                                delete Shifts->m_period;
                                Shifts->m_period = Pptr;
                        }
                        delete Shifts->m_period;
                }
                Sptr = Sptr->next;
                delete Shifts;
                Shifts = Sptr;
        }
        if (Shifts->m_period != NULL)
        {
                Pptr = Shifts->m_period;
                while (Shifts->m_period->next != NULL)
                {
                        Pptr = Pptr->next;
                        delete Shifts->m_period;
                        Shifts->m_period = Pptr;
                }
                delete Shifts->m_period;
        }
        delete Shifts;
}
update_value += 1.0f;
Progress.SetPos((int) update_value);
}
```

```
float CSolverDLL::GetNextTime(float curtime)
{
        float           nexttime;
        int             i,firsttime;
        CTelement       *Tptr;
        CRelement       *Rptr;
        CTimePer        *Pptr;
        CTimePer                        *nAptr;

        Rptr = ResList;
        nexttime = curtime+ELAPSE;
        while (Rptr != NULL)
        {
                if ((Rptr->m_avail_time > curtime) &&
                        (Rptr->m_avail_time < nexttime))
                        nexttime = Rptr->m_avail_time;
                if (Rptr->breaks != NULL)
                {
                        Pptr = Rptr->breaks->m_period;
                        while (Pptr != NULL)
                        {
                                if ((Pptr->m_start_tim > curtime) &&
                                        (Pptr->m_start_tim < nexttime))
                                        nexttime = Pptr->m_start_tim;
                                Pptr = Pptr->next;
                        }
                }
                Rptr = Rptr->next;
        }


        firsttime = FALSE;
        // assign any breaks if necessary
        Rptr = ResList;
        while (Rptr != NULL)
        {
                if (Rptr->breaks != NULL)
                {
                        Pptr = Rptr->breaks->m_period;
                        while (Pptr != NULL)
                        {
                                if ((Pptr->m_start_tim <= nexttime) &&
                                        (Pptr->m_end_tim > nexttime))
                                        if (Rptr->m_avail_time < Pptr->m_end_tim)
```

```
                                        {
                                                Rptr->m_avail_time = Pptr->m_end_tim;
                                                if (Pptr->next != NULL)
                                                        Rptr->m_not_avail =
Pptr->next->m_start_tim;
                                                else
                                                        Rptr->m_not_avail = Rptr->m_end_time;
                                                break;
                                        }
                                Pptr = Pptr->next;
                        }
                }
                if (Rptr->nAvail != NULL)
                {
                        nAptr = Rptr->nAvail;
                        while (nAptr != NULL)
                        {
                                if ((nAptr->m_start_tim <= nexttime) &&
                                        (nAptr->m_end_tim > nexttime))
                                        if (Rptr->m_avail_time < nAptr->m_end_tim)
                                        {
                                                Rptr->m_avail_time = nAptr->m_end_tim;
                                                if (nAptr->next != NULL)
                                                        Rptr->m_not_avail =
nAptr->next->m_start_tim;
                                                else
                                                        Rptr->m_not_avail = Rptr->m_end_time;
                                                break;
                                        }
                                nAptr = nAptr->next;
                        }
                }
                Rptr = Rptr->next;
        }

        // Introduce new queues into the system
        if (posEvent != NULL)
        {

                Tptr = posEvent;
                if (posEvent == EventList)
                {
                        curtime = posEvent->m_time;
                        nexttime = curtime + ELAPSE;
                        firsttime = TRUE;
```

```
                }
                while (Tptr != NULL)
                {
                        if (curtime >= Tptr->m_time)
                        {
                                i = 0;
                                while ((i < numTask) &&
                                        (QList[i].m_location_task_id !=
Tptr->m_location_task_id))
                                        i++;
                                if (QList[i].m_location_task_id == Tptr->m_location_task_id)
                                        QList[i].m_Rxs += Tptr->m_process_qty;
                        }
                        Tptr = Tptr->next;
                }
                posEvent = Tptr;
        }


        if (firsttime)
                return (__min(curtime,nexttime));
        else
                return (nexttime);
}


int CSolverDLL::AssignRes(int taskID, float curtime)
{
        CTRRes                  *Rptr;
        int                     done = FALSE;
        int                     cteam = 0;
        float           etime, min_rate;
        float                   tempdone;
        CAssign                 *oldptr;
        CMMList                 *mptr;
        CTeam                           *tptr;

        if (QList[taskID].teamList != NULL)
        {
                tptr = QList[taskID].teamList;
                while (tptr != NULL)
                {
                        cteam++;
                        tptr= tptr->next;
                }
```

```
if (QList[taskID].m_max_resource_qty >= (QList[taskID].m_assigned + cteam))
{
        etime = ELAPSE+curtime;
        tptr = QList[taskID].teamList;
        min_rate = 8888888.8f;
        while (tptr != NULL)
        {
                Rptr = QList[taskID].resList;
                done = FALSE;
                while ((Rptr != NULL) && (!done))
                {
                        if ((Rptr->resource->m_last_task ==
QList[taskID].m_location_task_id) &&
                                (Rptr->resource->m_resource_type_cde ==
tptr->m_resource_type_cde))
                        {
                                if ((Rptr->resource->m_avail_time <= curtime) &&
                                        (Rptr->resource->m_end_time > curtime)
&&
                                        (Rptr->resource->m_not_avail > curtime))
                                {
                                        mptr = QList[taskID].maxList;
                                        while (mptr != NULL)
                                        {
                                                if
(Rptr->resource->m_resource_type_cde == mptr->m_resource_type_cde)
                                                {
                                                        if (Rptr->m_total_time <
mptr->m_time_amt)
                                                                etime =
__min(etime,curtime+(mptr->m_time_amt - Rptr->m_total_time));
                                                        break;
                                                }
                                                mptr = mptr->next;
                                        }
                                        oldptr = Rptr->assigned;
                                        while ((oldptr != NULL) && (!done))
                                        {
                                                if (oldptr->m_end_tim == curtime)
                                                {
                                                        etime = __min(etime,
Rptr->resource->m_not_avail);
                                                        done = TRUE;
                                                        tptr->rptr = Rptr;
                                                        tptr->cont_prev = TRUE;
```

```
                                                    min_rate = __min(min_rate,
Rptr->m_hourly_rate);
                                                }
                                                oldptr = oldptr->next;
                                            }
                                        }
                                    }
                                    Rptr = Rptr->next;
                                }
                                Rptr = QList[taskID].resList;
                                while ((Rptr != NULL) && (!done))
                                {
                                    if ((Rptr->resource->m_resource_type_cde ==
tptr->m_resource_type_cde) &&
                                        (Rptr->resource->m_avail_time <= curtime) &&
                                        (Rptr->resource->m_end_time > curtime))
                                    {
                                        mptr = QList[taskID].maxList;
                                        while (mptr != NULL)
                                        {
                                            if (Rptr->resource->m_resource_type_cde
== mptr->m_resource_type_cde)
                                            {
                                                if (Rptr->m_total_time <
mptr->m_time_amt)
                                                    etime = __min(etime,curtime
+(mptr->m_time_amt - Rptr->m_total_time));
                                                break;
                                            }
                                            mptr = mptr->next;
                                        }
                                        if (etime <= Rptr->resource->m_not_avail)
                                        {
                                            done = TRUE;
                                            tptr->rptr = Rptr;
                                            tptr->cont_prev = FALSE;
                                            min_rate = __min(min_rate,
Rptr->m_hourly_rate);
                                        }
                                    }
                                    Rptr = Rptr->next;
                                }
                                if (!done)
                                    return(done);
                                else
```

```
                    tptr = tptr->next;
        }

        // do actual assignments
        tempdone = min_rate*(etime-curtime);
        if (tempdone > QList[taskID].m_Rxs)
        {
                tempdone = QList[taskID].m_Rxs;
        }
        tptr = QList[taskID].teamList;
        while (tptr != NULL)
        {
                if (tptr->cont_prev == TRUE)
                {
                        oldptr = tptr->rptr->assigned;
                        while (oldptr != NULL)
                        {
                                if (oldptr->m_end_tim == curtime)
                                {
                                        oldptr->m_end_tim = etime;
                                        oldptr->m_Rxs_done += tempdone;
                                        tptr->rptr->m_total_time +=(etime-curtime);
                                        tptr->rptr->resource->m_avail_time = etime;
                                        tptr->rptr->resource->m_calculate = TRUE;
                                        QList[taskID].m_assigned ++;
                                        break;
                                }
                                oldptr = oldptr->next;
                        }
                }
                else
                {
                        CAssign *Aptr = new CAssign();
                        Aptr->next = NULL;
                        Aptr->m_start_tim = curtime;
                        Aptr->m_Rxs_done = tempdone;
                        Aptr->m_end_tim = etime;
                        Aptr->m_Rxs_done = tempdone;
                        tptr->rptr->m_total_time += (etime - curtime);
                        tptr->rptr->resource->m_avail_time = etime;
                        tptr->rptr->resource->m_last_task =
QList[taskID].m_location_task_id;
                        QList[taskID].m_assigned ++;
                        tptr->rptr->resource->m_calculate = TRUE;
                        if (tptr->rptr->assigned == NULL)
```

```
                                        tptr->rptr->assigned = Aptr;
                                    else
                                    {
                                        oldptr = tptr->rptr->assigned;
                                        while (oldptr->next != NULL)
                                            oldptr = oldptr->next;
                                        oldptr->next = Aptr;
                                    }
                                }
                                tptr = tptr->next;
                            }
                            QList[taskID].m_done += tempdone;
                            QList[taskID].m_Rxs -= tempdone;
                        }
                    }
                    else
                    {
                        Rptr = QList[taskID].resList;
                        while ((Rptr != NULL) && (!done))
                        {
                            if (Rptr->resource->m_last_task == QList[taskID].m_location_task_id)
                            {
                                if ((Rptr->resource->m_avail_time <= curtime) &&
                                    (Rptr->resource->m_end_time > curtime))
                                {
                                    mptr = QList[taskID].maxList;
                                    while (mptr != NULL)
                                    {
                                        if (Rptr->resource->m_resource_type_cde ==
mptr->m_resource_type_cde)
                                        {
                                            if (Rptr->m_total_time <
mptr->m_time_amt)
                                                etime = mptr->m_time_amt -
Rptr->m_total_time;
                                            else
                                                etime = 0.0f;
                                            break;
                                        }
                                        mptr = mptr->next;
                                    }
                                    if (mptr == NULL)
                                        etime = ELAPSE;
                                    oldptr = Rptr->assigned;
                                    while ((oldptr != NULL) && (!done))
```

```
                                        {
                                                if ((oldptr->m_end_tim == curtime) && (etime >
0.0f))
                                                {
                                                        etime = __min(ELAPSE, etime);
                                                        etime =
__min(etime,Rptr->resource->m_not_avail - curtime);
                                                        tempdone = Rptr->m_hourly_rate*etime;
                                                        if (tempdone >= QList[taskID].m_Rxs)
                                                        {
        //                                                  etime =
QList[taskID].m_Rxs/Rptr->m_hourly_rate;
        //                                                  etime =
__min(etime,((int)(etime/(float)LOOKSET)+1)*(float)LOOKSET);
                                                                tempdone = QList[taskID].m_Rxs;
                                                        }
                                                        oldptr->m_end_tim = curtime + etime;
                                                        oldptr->m_Rxs_done += tempdone;
                                                        QList[taskID].m_done += tempdone;
                                                        QList[taskID].m_Rxs -= tempdone;
                                                        Rptr->m_total_time += etime;
                                                        Rptr->resource->m_avail_time =
curtime+etime;
                                                        QList[taskID].m_assigned ++;
                                                        Rptr->resource->m_calculate = TRUE;
                                                        done = TRUE;
                                                }
                                                oldptr = oldptr->next;
                                        }
                                }
                        }
                        Rptr = Rptr->next;
                }

                // If there was no resource which was assigned to this task last time
                Rptr = QList[taskID].resList;
                while ((Rptr != NULL) && (!done))
                {
                        if ((Rptr->resource->m_avail_time <= curtime) &&
                                (Rptr->resource->m_end_time > curtime))
                        {
                                mptr = QList[taskID].maxList;
                                while (mptr != NULL)
                                {
```

```
                    if (Rptr->resource->m_resource_type_cde ==
mptr->m_resource_type_cde)
                    {
                        if (Rptr->m_total_time < mptr->m_time_amt)
                            etime = mptr->m_time_amt -
Rptr->m_total_time;
                        else
                            etime = 0.0f;
                        break;
                    }
                    mptr = mptr->next;
                }
                if (mptr == NULL)
                    etime = ELAPSE;
                if (((curtime+ELAPSE) <=Rptr->resource->m_not_avail) &&
                    (etime > 0.0f))
                {
                    etime = __min(ELAPSE,etime);
                    tempdone = Rptr->m_hourly_rate*etime;

                    CAssign *Aptr = new CAssign();

                    Aptr->m_start_tim = curtime;
                    Aptr->m_Rxs_done = 0.0f;
                    Aptr->next = NULL;
                    if (Rptr->assigned == NULL)
                        Rptr->assigned = Aptr;
                    else
                    {
                        oldptr = Rptr->assigned;
                        while (oldptr->next != NULL)
                            oldptr = oldptr->next;
                        oldptr->next = Aptr;
                    }
                    if (tempdone >= QList[taskID].m_Rxs)
                    {
//                      etime = QList[taskID].m_Rxs/Rptr->m_hourly_rate;
                        tempdone = QList[taskID].m_Rxs;
                    }
                    Aptr->m_end_tim = curtime+etime;
                    Aptr->m_Rxs_done += tempdone;
                    QList[taskID].m_done += tempdone;
                    QList[taskID].m_Rxs -= tempdone;
                    Rptr->m_total_time += etime;
                    Rptr->resource->m_avail_time = curtime+etime;
```

```
                                        Rptr->resource->m_last_task =
QList[taskID].m_location_task_id;
                                        QList[taskID].m_assigned ++;
                                        Rptr->resource->m_calculate = TRUE;
                                        done = TRUE;
                                }
                        }
                        Rptr = Rptr->next;
                }
        }
        return(done);
}

void CSolverDLL::CalculateQ(float curtime)
{
        CToList         *tptr;
        int               i, j;

        for (i = 0; i < numTask; i++)
                if (QList[i].m_done > 0)
                {
                        if (QList[i].flowto != NULL)
                        {
                                tptr = QList[i].flowto;
                                while (tptr != NULL)
                                {
                                        j = 0;
                                        while ((j < numTask) &&
                                                (QList[j].m_location_task_id !=
tptr->m_to_task_id))
                                                j++;
                                        if (QList[j].m_location_task_id == tptr->m_to_task_id)
                                                QList[j].m_Rxs += QList[i].m_done *
tptr->m_allocation_pct;
                                        tptr = tptr->next;
                                }
                        }
                        QList[i].m_RealQ += QList[i].m_done;
                }
}


float CSolverDLL::ConvertDBTime(int temp)
{
        double   temp2;
```

```
        int             hrs;
        double  mins;
        float           temp4;

        temp2 = (temp/100.0f);
        hrs = (int) temp2;
        mins = (((temp2 - (double)hrs)*100.0f)/60.0f);
        temp4 = (float)(hrs + mins);
        return(temp4);

}

void CSolverDLL::SaveAssign()
{
        CString buffer;
        CAssign *Aptr;
        CTRRes  *Rptr;
        CSaveSet                Assignments(NULL);
        float           how_much;

#ifdef _DEBUG
        char*     pFileName = _T("test.dat");

        CFile rfile( pFileName, CFile::modeCreate | CFile::modeWrite );
        buffer.Format("Resource ID\tResource Type\tTask ID\tStart\tEnd\tLabor
Cost\tRate\tPieces in Period\n");
        rfile.Write(buffer, buffer.GetLength());
#endif

        Progress.AddtoList("Saving Assignment Information...");
        Assignments.m_ConnectStr = DB_CONNECT;
        try
        {
                Assignments.Open();
        }
        catch(CDBException *e)
        {
                e->ReportError(MB_ICONEXCLAMATION);
                CleanupMem();
                exit;
        }

        for (int i = 0; i < numTask; i++)
        {
                Rptr = QList[i].resList;
```

```
                while (Rptr != NULL)
                {
                        Aptr = Rptr->assigned;
                        while (Aptr != NULL)
                        {

#ifdef _DEBUG
                                buffer.Format("%d \t%d \t%d \t%s \t%s \t%.2f \t%f \t%.2f\n",
                                        Rptr->resource->m_resource_id,
                                        Rptr->resource->m_resource_type_cde,
                                        QList[i].m_location_task_id,
                                        UnConvertTime(Aptr->m_start_tim),
                                        UnConvertTime(Aptr->m_end_tim),
                                        (Aptr->m_end_tim -
Aptr->m_start_tim)*Rptr->resource->m_cost,
                                        Rptr->m_hourly_rate, Aptr->m_Rxs_done);
                                rfile.Write(buffer, buffer.GetLength());
#endif

                                if (Assignments.CanAppend() >0)
                                {
                                        Assignments.AddNew();
                                        Assignments.m_location_task_id =
QList[i].m_location_task_id;
                                        Assignments.m_resource_id =
Rptr->resource->m_resource_id;
                                        Assignments.m_scenario_id = SCENARIO;
                                        Assignments.m_start_time =
UnConvertTime1(Aptr->m_start_tim);
                                        Assignments.m_end_time =
UnConvertTime1(Aptr->m_end_tim);
                                        Assignments.Update();
                                }
                                Aptr = Aptr->next;
                        }
                        Rptr = Rptr->next;
                }
                update_value += (20.0f/numTask);
                Progress.SetPos((int)update_value);
        }

#ifdef _DEBUG
        rfile.Close();
#endif
```

```
        Assignments.Close();

        Progress.AddtoList("Saving the Queue Information...");
        how_much = update_value+ 20.0f;
        CGetQ          SaveQ(NULL);
        CQResultList   *qrptr;

#ifdef _DEBUG
        char*     pFileName1 = _T("test1.dat");

        CFile rf( pFileName1, CFile::modeCreate | CFile::modeWrite );
        buffer.Format("Start Time\tTask ID\tQueue at Start\tProcessed\n");
        rf.Write(buffer, buffer.GetLength());
#endif

        SaveQ.m_ConnectStr = DB_CONNECT;
        try
        {
                SaveQ.Open();
        }
        catch(CDBException *e)
        {
                e->ReportError(MB_ICONEXCLAMATION);
                CleanupMem();
                exit;
        }

        qrptr = resultQ;
        while (qrptr != NULL)
        {

#ifdef _DEBUG
                buffer.Format("%s \t%d \t%f \t%f \t%d\n",
                        UnConvertTime(qrptr->m_start_time),
                        qrptr->m_location_task_id,
                        qrptr->m_queue, qrptr->m_cum_processed,
                        qrptr->m_num_assigned);
                rf.Write(buffer, buffer.GetLength());
#endif

                if (SaveQ.CanAppend() >0)
                {
                        SaveQ.AddNew();
                        SaveQ.m_scenario_id = SCENARIO;
                        SaveQ.m_location_task_id = qrptr->m_location_task_id;
```

```
                        SaveQ.m_start_time = UnConvertTime1(qrptr->m_start_time);
                        SaveQ.m_queue = (long) qrptr->m_queue;
                        SaveQ.m_cum_processed = (long) qrptr->m_cum_processed;
/*                      CString temp;
                        temp.Format("float: queue %f cum %f int: queue %d cum %d",
                                qrptr->m_queue, qrptr->m_cum_processed, SaveQ.m_queue,
                                SaveQ.m_cum_processed);
                        MessageBox(buffer, "MARK",MB_OK);
*/                      SaveQ.m_num_assigned = qrptr->m_num_assigned;
                        SaveQ.Update();
                }
                qrptr = qrptr->next;
                update_value += (20.0f/2500);
                update_value = __min(how_much,update_value);
                Progress.SetPos((int) update_value);
        }
        SaveQ.Close();
        update_value = __max(how_much, update_value);
#ifdef _DEBUG
        rf.Close();
#endif
}

CString CSolverDLL::UnConvertTime(float tmp)
{
        CString buffer;
        int     hrs, min;

        hrs = (int) tmp;
        min = (int)((tmp - hrs)*60);
        buffer.Format("%.2d:%.2d", hrs, min);

        return(buffer);
}

int CSolverDLL::UnConvertTime1(float tmp)
{
        int     hrs, min;

        hrs = (int) tmp;
        min = (int)((tmp - hrs)*60);
        hrs = hrs*100;

        return(hrs+min);
}
```

```
void CSolverDLL::PrintQs(float curtime)
{
	int i;

	for (i = 0; i < numTask; i++)
	{
		CQResultList *qptr = new CQResultList();
		qptr->m_location_task_id = QList[i].m_location_task_id;
		qptr->m_cum_processed = QList[i].m_RealQ;
		qptr->m_queue = QList[i].m_Rxs;
		qptr->m_start_time = curtime;
		qptr->m_num_assigned = QList[i].m_assigned;
		qptr->next = NULL;
		if (resultQ != NULL)
			posQList->next = qptr;
		else
			resultQ = qptr;
		posQList = qptr;
	}
}

void CSolverDLL::CalculateWork(float curtime)
{
	CTRRes		*Rptr;
	float		 AvgRate;

	// Calculate the amount of work which is left in each task
	for (int i = 0; i < numTask; i++)
	{
		if ((QList[i].m_Rxs > 0) &&
			(QList[i].m_start_tim <= (curtime)) &&
			(QList[i].m_end_tim > curtime))
		{
			AvgRate = 0.0f;
			Rptr = QList[i].resList;
			if (Rptr != NULL)
			{
				while (Rptr != NULL)
				{
					if ((Rptr->resource->m_avail_time <= curtime) &&
						(Rptr->resource->m_end_time > curtime))
					{
						if (AvgRate == 0)
							AvgRate = Rptr->m_hourly_rate;
						else
```

```
                                                AvgRate =
(AvgRate+Rptr->m_hourly_rate)/2;
                                        }
                                        Rptr = Rptr->next;
                                }
                                if (AvgRate == 0)
                                        QList[i].m_work = 0.0f;
                                else
                                        QList[i].m_work = QList[i].m_Rxs/AvgRate;
                        }
                        else
                                QList[i].m_work = 0.0f;
                }
                else
                        QList[i].m_work = 0.0f;
        }
}

int CSolverDLL::GetNextTask(float CurTime)
{
        for (int i = 0; i < numTask; i++)
                if ((QList[i].m_assigned < QList[i].m_max_resource_qty) &&
                        (QList[i].m_start_tim <= CurTime) &&
                        (QList[i].m_end_tim > CurTime) &&
                        (QList[i].m_go == 0) &&
//                          Assigns based on the Rxs
                        (QList[i].m_Rxs > 1) &&
//                          Assigns based on the amount of work left- must be ElapseTime
//                  (QList[i].m_work >= ELAPSE) &&
                        (QList[i].resList != NULL))
                        return(i);
        return(numTask);
}


int CSolverDLL::gcd(int u, int v)
{
        int t;
        while (u > 0)
        {
                if (u < v)
                {
                        t = u;
                        u = v;
                        v = t;
```

```
            }
            u = u - v;
        }
        return(v);
}


void CSolverDLL::GetEventList()
{
  /* Gets a list of the incoming queues and places it into
            the EventList  */
        CQueueSet    Incoming(NULL);
        CResultQ                    StartQ(NULL);

        CTelement     *cTptr;

        if (OLD_SCENARIO > 0)
        {
            Progress.AddtoList("Getting Queues for Re-Run of Scenario...");
            StartQ.m_ConnectStr = DB_CONNECT;
            StartQ.m_strSort = _T("location_task_id");
            StartQ.m_Scenario_Param = OLD_SCENARIO;
            StartQ.m_Start_Param = UnConvertTime1(STARTING);
            StartQ.Open();
            while (!StartQ.IsEOF())
            {
                int i = 0;
                while (QList[i].m_location_task_id != StartQ.m_location_task_id)
                    i++;
                if (QList[i].m_location_task_id == StartQ.m_location_task_id)
                    QList[i].m_Rxs = (float)StartQ.m_queue;
                StartQ.MoveNext();
            }
            StartQ.Close();
        }

        update_value += 1.0f;
        Progress.SetPos((int) update_value);
        Progress.AddtoList("Getting Queues...");
        Incoming.m_ConnectStr = DB_CONNECT;
        Incoming.m_strSort = _T("start_tim");
        Incoming.m_strFilter = _T("(tq.location_task_id = lt.location_task_id) \
            and (lt.location_nbr = ?) and (start_tim >= ?)");
        Incoming.m_Pharmacy_Param = PHARMACY;
        Incoming.m_Start_Param = UnConvertTime1(STARTING);
```

```
        try
        {
                Incoming.Open();
        }
        catch(CDBException *e)
        {
                e->ReportError(MB_ICONEXCLAMATION);
                CleanupMem();
                exit;
        }

        while (!Incoming.IsEOF())
        {
                CTelement *Tptr = new CTelement();
                Tptr->m_time = ConvertDBTime(Incoming.m_start_tim);
                Tptr->m_location_task_id = Incoming.m_location_task_id;
                Tptr->m_process_qty = (float)Incoming.m_process_qty;
                Tptr->next = NULL;
                if (EventList == NULL)
                        EventList = Tptr;
                else
                        cTptr->next = Tptr;
                cTptr = Tptr;
                Incoming.MoveNext();
        }
        Incoming.Close();
        posEvent = EventList;
        update_value += 1.0f;
        Progress.SetPos((int) update_value);

}


extern "C" int FAR PASCAL EXPORT
FastSolve(LPCTSTR connect_str, LPCTSTR sday_cde, LPCTSTR pharm_cde,
                        int elapse_min, int scen_id, int old_id, int start_time,
                        int break_id)
{
   // AFX_MANAGE_STATE(AfxGetStaticModuleState());

        CSolverDLL    goDLL;
        CString                 buffer;
        CTime                           day_week;
        int                     week_day;
        CRCount                 GetTaskCount(NULL);
```

```
        CString day_cde(sday_cde);


//      MessageBox(NULL,"Make Green Screen","Mark",MB_OK);
        goDLL.Progress.Create();
        goDLL.Progress.SetPos(0);
        goDLL.Progress.AddtoList("Getting the number tasks in system");

        day_week = CTime(atoi(day_cde.Mid(6,4)), atoi(day_cde.Mid(0,2)),
                atoi(day_cde.Mid(3,2)),0,0,0);
        week_day = day_week.GetDayOfWeek();
        if (week_day == 1)
                week_day = 7;
        else
                week_day--;

/*#ifdef _DEBUG
        MessageBox(NULL,connect_str,"Mark",MB_OK);
        MessageBox(NULL,day_cde,"Mark",MB_OK);
        MessageBox(NULL,pharm_cde,"Mark",MB_OK);
        buffer.Format("%d",elapse_min);
        MessageBox(NULL,buffer,"Mark",MB_OK);
        buffer.Format("%d",scen_id);
        MessageBox(NULL,buffer,"Mark",MB_OK);
        buffer.Format("%d",old_id);
        MessageBox(NULL,buffer,"Mark",MB_OK);
        buffer.Format("%d",start_time);
        MessageBox(NULL,buffer,"Mark",MB_OK);
#endif
*/
        GetTaskCount.m_ConnectStr = connect_str;
        GetTaskCount.m_day_Param.Format("%d",week_day);
        GetTaskCount.m_Pharmacy_Param = pharm_cde;
        try
        {
                GetTaskCount.Open();
//              MessageBox(NULL,"Open Task Count - 1","Mark",MB_OK);
        }
        catch(CDBException *e)
        {
                e->ReportError(MB_ICONEXCLAMATION);
                exit;
        }
```

```
        if (!GetTaskCount.IsEOF())
        {
                goDLL.update_value = 0.0f;
                goDLL.numTask = GetTaskCount.m_count;
                goDLL.QList = new CQelement[goDLL.numTask];
                for (int i = 0; i < goDLL.numTask; i++)
                {
                        goDLL.QList[i].resList = NULL;
                        goDLL.QList[i].flowto = NULL;
                        goDLL.QList[i].ratList = NULL;
                        goDLL.QList[i].teamList = NULL;
                        goDLL.QList[i].minList = NULL;
                        goDLL.QList[i].maxList = NULL;
                }
                goDLL.BREAKID = break_id;
                goDLL.RUN_DATE_TIME = day_week;
                goDLL.DB_CONNECT = connect_str;
                goDLL.DAYCDE.Format("%d",week_day);
                goDLL.OLD_SCENARIO = old_id;
                goDLL.SCENARIO = scen_id;
                goDLL.STARTING = goDLL.ConvertDBTime(start_time);
                goDLL.PHARMACY = pharm_cde;
                goDLL.ELAPSE =  (float)(elapse_min/60.0);
//              MessageBox(NULL,"Go into the Algorithm - 2","Mark",MB_OK);
                goDLL.OnRun();
        }
        GetTaskCount.Close();
        goDLL.Progress.AddtoList("Done...");
        goDLL.Progress.SetPos(100);
        goDLL.Progress.DestroyWindow();
        return (1);
}
```